修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　　情報理工学研究科　情報・ネットワーク工学専攻　博士前期課程		
氏　　名	荒井 嵩博	学籍番号	1931005
論 文 題 目	LWE 暗号における IND-CPA 安全性の再評価		

要　旨

現在幅広く運用されている RSA 暗号などの暗号方式は、従来のコンピュータで素因数分解や離散対数問題を解くのに非常に長い時間がかかるということを安全性の根拠としている。しかし、Shor が発表した素因数分解アルゴリズムや離散対数問題のアルゴリズムは量子コンピュータを用いると従来より短い時間でこれらの問題を解くことが可能になった。つまり、量子コンピュータの実用化すると、現在使われている公開鍵暗号は安全性を保障できなくなる。

そこで近年では、量子コンピュータに対しても安全性を保障できる耐量子暗号の研究が活発化しており、符号暗号や多変数多項式を用いた暗号、そして本研究で用いた格子暗号が耐量子暗号として盛んに研究されている。格子暗号とは、ある行列がつくる格子について最短ベクトルを求める問題(Shortest Vector Problem. 以下 SVP) の困難性などを利用した暗号である。本研究で用いた Learning with Errors 問題をもとにした暗号(以下 LWE 暗号) も格子暗号の 1 つである。LWE 暗号は、2005 年に Regev によって発表され、現在はこの暗号をもとに多くの対量子暗号方式が開発されている。

LWE 暗号の欠点として鍵の長さがかなり長くないと安全性を保障できないという点が挙げられる。2017 年、Liu らに提案された Compact-LWE は、LWE 暗号より鍵の長さを短くしても安全性が保障できると予測された暗号方式である。しかし、2018 年に Bootle らによって攻撃手法が提案された。

本論文では、平文 1 ビットにおける IND-CPA 安全性と等価性を持つより単純な安全性定義を提案し、その安全性に対する新たな LWE 暗号への攻撃手法を Bootle らの攻撃手法を LWE 暗号へ拡張することで構成した。計算機実験では、lwe-estimator により生成した、30 ビット安全性を保証可能と想定されるパラメータにおける LWE 暗号に対し提案攻撃法を適用した。その結果、攻撃法の計算量は$\mathcal{O}(2^{40})$ となったことから、実験で用いたパラメータは我々の提案攻撃法に対して安全であることが判明した。

さらに、lwe-estimator により生成した、31 ビット、32 ビット安全性を保証可能と想定されるパラメータでも同様の実験を行い、計算量を導出したところ、それぞれ$\mathcal{O}(2^{47})$、$\mathcal{O}(2^{54})$となった。このことから、31 ビット、32 ビット安全性を保証可能と想定されるパラメータでも、我々の提案攻撃法に対して安全であることが判明した。

令和２年度　修士学位論文

LWE暗号におけるIND-CPA安全性の再評価

学籍番号　1931005

氏　　名　荒井 嵩博

指導教員　バグス サントソ 准教授

大濱　靖匡 教授

川端　勉 教授

電気通信大学 情報理工学研究科

博士前期課程 情報・ネットワーク工学専攻専攻

提出　令和３年１月25日

THE UNIVERSITY OF ELECTRO-COMMUNICATIONS
Department of Communications and Systems Engineering

目 次

第 1 章

はじめに

2019 年 1 月 8 日、アメリカ ラスベガスで行われた CES で IBM が世界初となる商用量子コンピュータが発表された。まだ、小規模なものであるが、将来の実用化に向けての第一歩である。

現在幅広く運用されている RSA 暗号などの暗号方式は、従来のコンピュータで素因数分解や離散対数問題を解くのに非常に長い時間がかかるということを安全性の根拠としている。しかし、Shor が発表した素因数分解アルゴリズムや離散対数問題のアルゴリズム [3] は量子コンピュータを用いると従来より短い時間でこれらの問題を解くことが可能になった。つまり、量子コンピュータの実用化すると、現在使われている公開鍵暗号は安全性を保障できなくなる。

そこで近年では、量子コンピュータに対しても安全性を保障できる耐量子暗号の研究が活発化しており、符号暗号や多変数多項式を用いた暗号、そして本研究で用いた格子暗号が耐量子暗号として盛んに研究されている。格子暗号とは、ある行列がつくる格子について最短ベクトルを求める問題 (Shortest Vector Problem. 以下 SVP) の困難性などを利用した暗号である。本研究で用いた Learning with Errors 問題をもとにした暗号 (以下 LWE 暗号 [4]) も格子暗号の 1 つである。LWE 暗号は、2005 年に Regev によって発表され、現在はこの暗号をもとに多くの対量子暗号方式が開発されている。

LWE 暗号の欠点として鍵の長さがかなり長くないと安全性を保障できないという点が挙げられる。2017 年、Liu らに提案された Compact-LWE[1] は、LWE 暗号より鍵の長さを短くしても安全性が保障できると予測された暗号方式である。しかし、2018 年に Bootle らによって攻撃手法が提案された [2]。

本論文では、平文 1 ビットにおける IND-CPA 安全性と等価性を持つより単純な安全性定義を提案し、その安全性に対する新たな LWE 暗号への攻撃手法を Bootle らの攻撃手法を LWE 暗号へ拡張することで構成した。計算機実験では、lwe-estimator[5] により生成した、30 ビット安全性を保証可能と想定されるパラメータにおける LWE 暗号に対し提案攻撃法を適用した。その結果、格子簡約アルゴリズムの計算量を T とすれば、提案攻撃法の計算量は $O(2^{23}) \times T$ になることが判明した。また文献 [5] を参考に具体的な T

を導出し上記の結果に代入したところ、攻撃法の計算量は $O(2^{40})$ となったことから、実験で用いたパラメータは我々の提案攻撃法に対して安全であることが判明した。さらに、lwe-estimator により生成した、31 ビット、32 ビット安全性を保証可能と想定されるパラメータでも同様の実験を行い、計算量を導出したところ、それぞれ $O(2^{47})$、$O(2^{54})$ となった。このことから、31 ビット、32 ビット安全性を保証可能と想定されるパラメータでも、我々の提案攻撃法に対して安全であることが判明した。

本論文の構成は、2 章で準備として表記の説明や公開鍵暗号、IND-CPA 安全などの前提知識を述べる。3 章で今回新たに定義した 1bit 平文暗号方式に対する安全性 (IND-1bit 安全) とその定義と IND-CPA 安全の等価性に必要な定理を述べ、4 章でその定理の証明を述べる。5 章では、IND-1bit を実際に計算機実験に使えるよう具体化した補題とその証明を述べる。6 章で実際に構成した攻撃手法について述べ、7 章で 30 ビット安全性を保証可能と想定されるパラメータに対し実験を行い LWE 暗号の安全性の評価を行った。さらに、8 章で 31 ビット、32 ビット安全性を保証可能と想定されるパラメータに対しても実験を行い LWE 暗号の安全性の評価を行った。

第 2 章

準備

2.1 表記

アルゴリズムを $\mathcal{A}$、集合を S、分布を D とするとき、$\leftarrow$ は以下の意味を持つとする。

· $y \leftarrow \mathcal{A}(x)$: y は、アルゴリズム $\mathcal{A}$ に x とランダムテープを入力して生成された値である。

· $y \leftarrow \$S$: y は、集合 S から一様分布に従って生成された値である。

· $y \leftarrow D$: y は、分布 D に従って生成された値である。

また、$\mathbb{Z}_q$ は 0 から $q-1$ までの整数の集合である。

2.2 公開鍵暗号

セキュリティパラメータを 1^k、公開鍵を pk、秘密鍵を sk とするとき、$(pk, sk) \leftarrow \mathrm{Keygen}(1^k)$ となる Keygen を鍵生成アルゴリズムとする。また、平文を v、暗号文を c とするとき、$c \leftarrow \mathrm{Enc}(pk, v)$ となる Enc を暗号化アルゴリズム、$v \leftarrow \mathrm{Dec}(sk, c)$ となる Dec を復号アルゴリズムとする。このとき、$\mathcal{E} = (\mathrm{Keygen}, \mathrm{Enc}, \mathrm{Dec})$ を公開鍵暗号方式と呼ぶ。本論文では、暗号方式は全て公開鍵暗号方式である。

2.3 IND-CPA 安全

IND-CPA 安全を定義するために IND-CPA ゲームを示す。

定義 2.3.1 (IND-CPA ゲーム)
ある暗号方式 $\mathcal{E}$ において、攻撃者と挑戦者間で以下のようなゲームを考える。これを *IND-CPA* ゲームという。

1. 攻撃者は挑戦者から pk を受け取る。

2. 攻撃者は同じ長さで違う値の平文 (v_0, v_1) を用意し、挑戦者にわたす。
3. 挑戦者はビット $d \in \{0,1\}$ をランダムに選び、暗号文 $c = \mathrm{Enc}(pk, v_d)$ を攻撃者にわたす。
4. 攻撃者は c から d を予想し d' として出力する。

定義 2.3.2 (IND-CPA 安全)
任意の *IND-CPA* ゲームの攻撃者 $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ が以下を満たすとき、暗号方式 $\mathcal{E}$ が *IND-CPA* 安全であるという。

$$\left| \Pr\left[d' = d \;\middle|\; \begin{array}{l} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_d), d' \leftarrow \mathcal{A}_1(c, st) \end{array} \right] - \frac{1}{2} \right| \leq \epsilon(k) \tag{2.1}$$

ここで st は内部情報である。

2.4　LWE 暗号

2.4.1　パラメータ

$pp = \{n, q, m, \sigma\}$ がパラメータである。ただし、n, q, m は正の整数、σ は正の実数である。

2.4.2　鍵生成

パラメータ pp から秘密鍵 sk, 公開鍵 pk を生成する。
$(sk, pk) \leftarrow \mathrm{KeyGen}(pp)$:

· **秘密鍵** $sk = \vec{s}$

$$\vec{s} \leftarrow \$\mathbb{Z}_q^n \tag{2.2}$$

· **公開鍵** $pk = \{\vec{a_i}, b_i\}_{1 \leq i \leq m}$

$$\vec{a_i} \leftarrow \$\mathbb{Z}_q^n,\ e_1, \cdots, e_m \leftarrow \chi,\ b_i = \langle \vec{a_i}, \vec{s} \rangle + e_i \bmod q \tag{2.3}$$

ここで χ は平均 0、標準偏差 σ の離散ガウス分布。

2.4.3　暗号化

pk、平文 $v \in \{0,1\}$ から暗号文 c を生成。
$c \leftarrow \mathrm{Enc}(pk, v)$:

$$
\begin{aligned}
&l_1, \cdots, l_m \leftarrow \$\{0,1\}, \\
&c = \left(\sum_{i=1}^{m} l_i \vec{a_i} \bmod q, v \left\lfloor \frac{q}{2} \right\rfloor + \sum_{i=1}^{m} l_i b_i \bmod q \right)
\end{aligned} \tag{2.4}
$$

2.4.4　復号

sk, c から v を生成。
$v \leftarrow \mathrm{Dec}(sk, c)$: $(\vec{a}, b) = c$ として $b - \langle \vec{a}, \vec{s} \rangle \bmod q$ が $\left\lfloor \frac{q}{2} \right\rfloor$ より 0 に近い値をとれば $v = 0$。それ以外ならば $v = 1$ を出力。

2.5　Compact-LWE[1]

Compact-LWE は LWE 暗号をもとにした暗号であり、LWE 暗号の鍵の長さが大きくなければ安全性を保証できないという欠点を改善する目的で 2017 年に提案された [1]。しかし、2018 年に攻撃手法が見つかったと発表された [2]。

2.5.1　パラメータ

パラメータ $pp = \{q, n, m, t, w, b\}$ は以下の条件を満たす。ただし、q, n, m, t, w, b は正の整数である。

$$
n + 1 < m < n^2,\ 2b(b \log_2 b + 1) < q,\ 2 \log_2 b < n \tag{2.5}
$$

2.5.2　鍵生成

パラメータ pp から秘密鍵 sk、公開鍵 pk を生成する。
$(sk, pk) \leftarrow \mathrm{KeyGen}(pp)$:

· **秘密鍵** $sk = (\vec{s}, r, p, h)$
　$\vec{s}, r, p, h$ を以下のように選ぶ。

$$
\vec{s} \leftarrow \$\mathbb{Z}_q^{n \times 1},\ r \in \mathbb{Z}_q,\ p \in \mathbb{Z}_q,\ h \in \mathbb{Z}_q \tag{2.6}
$$

ただし、以下の条件を満たす。

$$b < r,\ t \leq p,\ h(t-1) + wrp < q. \tag{2.7}$$

但し、

$$\gcd(h, q) = 1, \gcd(h, p) = 1, \gcd(p, q) = 1. \tag{2.8}$$

· **公開鍵** $pk = \{\vec{a_i}, \beta_i\}_{1 \leq i \leq m}$

$\vec{a_i}, e_1, \cdots, e_m$ を以下のように選ぶ。

$$\vec{a_i} \leftarrow \$\mathbb{Z}_b^{n \times 1},\ e_1, \cdots, e_m \leftarrow \$\mathbb{Z}_r \tag{2.9}$$

h_q^{-1} を以下のように定義し、

$$h_q^{-1} = (-h)^{-1} \bmod q\ \ (\because (2.8)) \tag{2.10}$$

β_i を生成する。

$$\beta_i = \langle \vec{a_i}, \vec{s} \rangle + h_q^{-1} p e_i \bmod q \tag{2.11}$$

2.5.3　暗号化

pk、平文 $v \in \mathbb{Z}_t$ から暗号文 c を生成。

$c \leftarrow \mathrm{Enc}(pk, v)$:

$$i_1, \cdots, i_w \leftarrow \$\{1, \cdots, m\} \tag{2.12}$$

$$\vec{a} = \sum_{k=1}^{w} \vec{a_{i_k}},\ \beta = \sum_{k=1}^{w} \beta_{i_k},\ c = (\vec{a}, v - \beta \bmod q) \tag{2.13}$$

2.5.4　復号

sk, c から v を生成。

$v \leftarrow \mathrm{Dec}(sk, c)$:

$$(\vec{c_0}, c_1) = c = (\vec{a}, v - \beta \bmod q) \tag{2.14}$$

$$v = -h_p^{-1}(h(\langle \vec{c_0}, \vec{s} \rangle + c_1) \bmod q) \bmod p \tag{2.15}$$

$$h_p^{-1} = (-h)^{-1} \bmod p\ \ (\because (2.8)) \tag{2.16}$$

第 3 章

1bit平文暗号方式に対する新たな安全性の定義

3.1 定義

平文が1bitである暗号方式の新たな安全性定義を示すため、IND-1bitゲームを示す。

定義 3.1.1 (IND-1bit ゲーム)
ある暗号方式 $\mathcal{E}$ において、攻撃者と挑戦者間で以下のようなゲームを考える。これを *IND-1bit* ゲームという。

1. 攻撃者は挑戦者から pk を受け取る。
2. 挑戦者は平文 $v \in \{0,1\}$ をランダムに選び、暗号文 $c = \mathrm{Enc}(pk, v)$ を攻撃者にわたす。
3. 攻撃者は c から v を予想し v' として出力する。

定義 3.1.2 (IND-1bit 安全)
任意の *IND-1bit* ゲームの攻撃者 $\mathcal{D}$ が以下を満たすとき、暗号方式 $\mathcal{E}$ が *IND-1bit* 安全であるという。

$$\left| \Pr\left[v' = v \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), v \leftarrow \$\{0,1\}, \\ c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(pk, c) \end{array} \right] - \frac{1}{2} \right| \leq \epsilon(k) \tag{3.1}$$

3.2 IND-1bit安全と1bit平文暗号方式におけるIND-CPA安全の等価性

本節では、等価性の証明に必要な定理とその系を示す。

3.2.1　IND-CPA 安全 ⇒IND-1bit 安全

定理 3.2.1

平文が *1bit* である暗号方式 $\mathcal{E}$ を考える。*IND-1bit* ゲームにおいてある攻撃者 $\mathcal{D}$ が以下を満たすとする。

$$\left|\Pr\left[v' = v \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), v \leftarrow \$\{0,1\}, \\ c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(pk, c)\end{array}\right] - \frac{1}{2}\right| > \epsilon(k) \tag{3.2}$$

このとき *IND-CPA* ゲームにおいてある攻撃者 $\mathcal{A}$ が以下を満たす。

$$\left|\Pr\left[d' = d \;\middle|\; \begin{array}{c}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_d), d' \leftarrow \mathcal{A}_1(c, st)\end{array}\right] - \frac{1}{2}\right| > \epsilon(k) \tag{3.3}$$

定理 3.2.1 の対偶をとると、以下の系が成り立つ。

系 3.2.1

平文が *1bit* の暗号方式 $\mathcal{E}$ が *IND-CPA* 安全を満たすとき *IND-1bit* 安全が成り立つ。

3.2.2　IND-1bit 安全 ⇒IND-CPA 安全

定理 3.2.2

平文が *1bit* である暗号方式 $\mathcal{E}$ を考える。*IND-CPA* ゲームにおいてある攻撃者 $\mathcal{A}$ が以下を満たすとする。

$$\left|\Pr\left[d' = d \;\middle|\; \begin{array}{c}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_d), d' \leftarrow \mathcal{A}_1(c, st)\end{array}\right] - \frac{1}{2}\right| > \epsilon(k) \tag{3.3}$$

このとき *IND-1bit* ゲームにおいてある攻撃者 $\mathcal{D}$ が以下を満たす。

$$\left|\Pr\left[v' = v \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), v \leftarrow \$\{0,1\}, \\ c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(pk, c)\end{array}\right] - \frac{1}{2}\right| > \epsilon(k) \tag{3.2}$$

定理 3.2.2 の対偶をとると、以下の系が成り立つ。

系 3.2.2

平文が *1bit* の暗号方式 $\mathcal{E}$ が *IND-1bit* 安全を満たすとき *IND-CPA* 安全が成り立つ。

系 3.2.1 と系 3.2.2 より、以下の系が成り立つ。

系 3.2.3

1bit 平文の暗号方式において *IND-CPA* 安全と *IND-1bit* 安全は等価である。

第 4 章

定理の証明

この章では、3 章で述べた定理の証明を示す。

4.1 定理 3.2.1 の証明

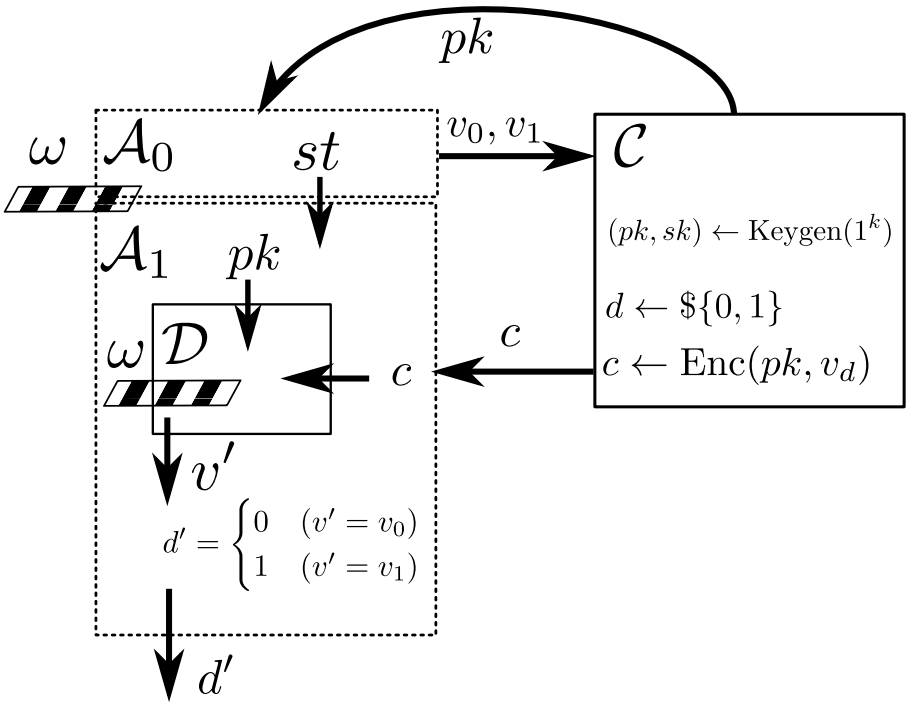

図 4.1: 定理 3.2.1 の証明

以下では IND-1bit の攻撃者 $\mathcal{D}$ から IND-CPA の攻撃者 $\mathcal{A}$ の構成方法を説明する。

1. 攻撃者 $\mathcal{A}_0$ は挑戦者 $\mathcal{C}$ から pk を受け取る。

2. 攻撃者 $\mathcal{A}_0$ は同じ長さで違う値の平文 $v_0 = 0, v_1 = 1$ を用意し、挑戦者 $\mathcal{C}$ にわたす。また、 内部情報 st を攻撃者 $\mathcal{A}_1$ にわたす。
3. 挑戦者 $\mathcal{C}$ はビット $d \in \{0,1\}$ をランダムに選び、暗号文 $c = \mathrm{Enc}(pk, v_d)$ を攻撃者 $\mathcal{A}_1$ にわたし、攻撃者 $\mathcal{A}_1$ はそれと pk を攻撃者 $\mathcal{D}$ にわたす。
4. 攻撃者 $\mathcal{D}$ が v' を出力する。
5. 攻撃者 $\mathcal{A}_1$ は $v' = v_0$ のときは $d' = 0$ を, $v' = v_1$ のときは $d' = 1$ を出力する。

上記の構成方法に基づいて、式 (3.3) の左辺を式変形していく。 d は手順 (2) で $v_d = d$ と決め、 $d \leftarrow \$\{0,1\}$ と $v \leftarrow \$\{0,1\}$ から $v = v_d = d$ と変換できる。また、 d' も手順 (5) で $v' = v_0 = 0$ のとき $d' = 0$, $v' = v_1 = 1$ のとき $d' = 1$ と決めたことから $v' = d'$ と変換できる。

$$
\begin{aligned}
&\left| \Pr\left[d' = d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_d), d' \leftarrow \mathcal{A}_1(c, st) \end{array} \right] - \frac{1}{2} \right| \\
&= \left| \Pr\left[v' = v \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ v \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(pk, c) \end{array} \right] - \frac{1}{2} \right| \qquad (4.1)
\end{aligned}
$$

ここで、$\mathcal{A}_0$ は図 4.1 の通り式 (4.1) の確率に影響を与えないので、

$$
(4.1) = \left| \Pr\left[v' = v \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), v \leftarrow \$\{0,1\}, \\ c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(pk, c) \end{array} \right] - \frac{1}{2} \right| > \epsilon(k)
$$

となり定理 3.2.1 が証明できた。 □

4.2　定理 3.2.2 の証明

攻撃者 $\mathcal{D}$ を $\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1)$ と作り、 入出力をそれぞれ $st \leftarrow \mathcal{D}_0(pk), v' \leftarrow \mathcal{D}_1(st, c)$ とする。

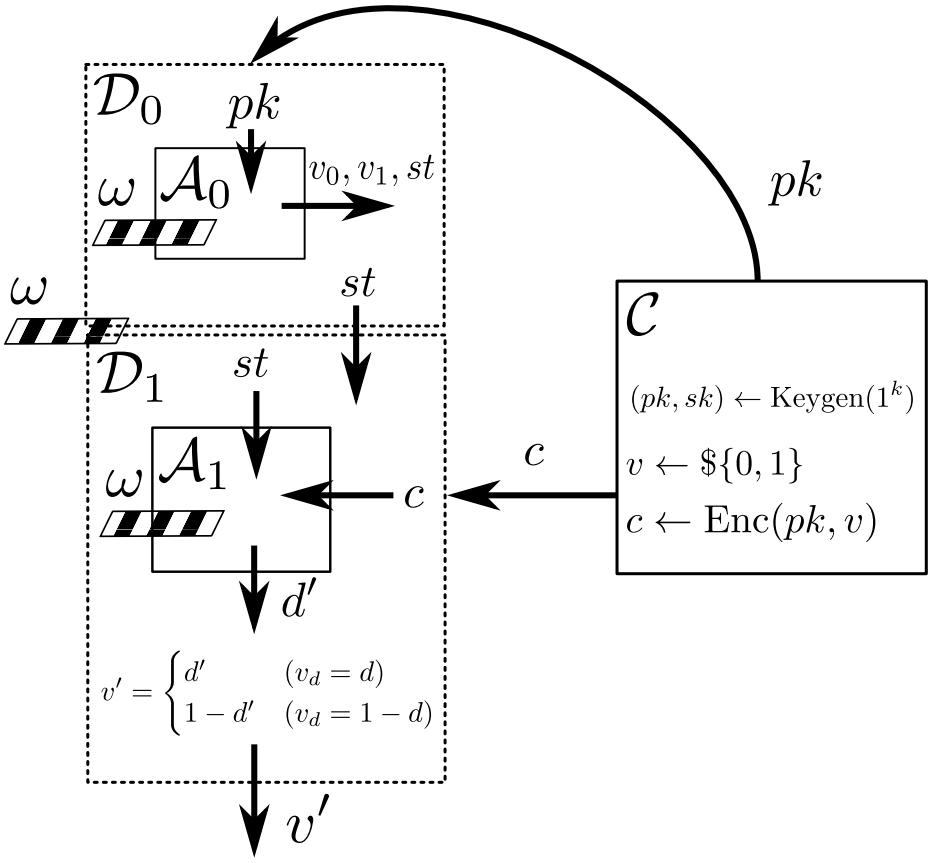

図 4.2: 定理 3.2.2 の証明

以下では IND-CPA の攻撃者 $\mathcal{A}$ から IND-1bit の攻撃者 $\mathcal{D}$ の構成方法を説明する。

1. 攻撃者 $\mathcal{D}_0$ は挑戦者 $\mathcal{C}$ から pk を受け取り、攻撃者 $\mathcal{A}_0$ にわたす。
2. 攻撃者 $\mathcal{A}_0$ が同じ長さで違う値の平文 $v_0, v_1 \in \{0,1\}$ と内部情報 st を出力する。
3. 攻撃者 $\mathcal{D}_0$ は内部情報 st を攻撃者 $\mathcal{D}_1$ にわたす。
4. 挑戦者 $\mathcal{C}$ は平文 $v \in \{0,1\}$ をランダムに選び、暗号文 $c = \mathrm{Enc}(pk, v)$ を攻撃者 $\mathcal{D}_1$ にわたし、攻撃者 $\mathcal{D}_1$ はそれと st を攻撃者 $\mathcal{A}_1$ にわたす。
5. 攻撃者 $\mathcal{A}_1$ が d' を出力する。
6. 攻撃者 $\mathcal{D}_1$ は $v' = v_{d'}$ となる v' を出力する。

式 (3.2) の左辺を式変形していく。$\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1)$ としたので変形して、

$$\begin{aligned} &\left| \Pr\left[v' = v \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), v \leftarrow \$\{0,1\}, \\ c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(c) \end{array} \right] - \frac{1}{2} \right| \\ &= \left| \Pr\left[v' = v \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), st \leftarrow \mathcal{D}_0(pk), \\ v \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \end{aligned} \tag{4.2}$$

ここで、(4.2) の式変形を $\mathcal{A}_0$ の出力が $v_d = d$ の場合と $v_d = 1 - d$ の場合に分けて考える。

$\mathcal{A}_0$ の出力が $v_d = d$ の場合、$d \leftarrow \$\{0,1\}, v \leftarrow \$\{0,1\}$ はともに $\mathcal{C}$ が決めるので $v = d$ とでき、$v_d = d$ のため $v = d = v_d$ と変換できる。

$$(4.2) = \left| \Pr \left[v' = d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), st \leftarrow \mathcal{D}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_d), v' \leftarrow \mathcal{D}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \tag{4.3}$$

また、$\mathcal{D}$ の出力を $v' = d'$ と決めると、

$$(4.3) = \left| \Pr \left[d' = d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), st \leftarrow \mathcal{D}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_d), d' \leftarrow \mathcal{D}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \tag{4.4}$$

$d' \leftarrow \mathcal{D}_1(st, c)$ は $d' \leftarrow \mathcal{A}_1(st, c)$ そのものであり、$\mathcal{D}_0, \mathcal{A}_0$ は式 (4.4) の確率に影響を与えないので、

$$(4.4) = \left| \Pr \left[d' = d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_d), d' \leftarrow \mathcal{A}_1(st, c) \end{array} \right] - \frac{1}{2} \right| > \epsilon(k)$$

となり $\mathcal{A}_0$ の出力が $v_d = d$ の場合は証明できた。

$\mathcal{A}_0$ の出力が $v_d = 1 - d$ の場合、$d \leftarrow \$\{0,1\}, v \leftarrow \$\{0,1\}$ はともに $\mathcal{C}$ が決めるので $v = d$ とでき、$v_d = 1 - d \rightarrow v_{1-d} = d$ のため $v = d = v_{1-d}$ と変換できる。

$$(4.2) = \left| \Pr \left[v' = d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), st \leftarrow \mathcal{D}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_{1-d}), v' \leftarrow \mathcal{D}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \tag{4.5}$$

また、$\mathcal{D}_1$ の出力を $v' = 1 - d'$ と決めると、

$$(4.5) = \left| \Pr \left[1 - d' = d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), st \leftarrow \mathcal{D}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_{1-d}), 1 - d' \leftarrow \mathcal{D}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \tag{4.6}$$

$\mathcal{D}_1$ は $\mathcal{A}_1$ の出力を反転して出力すると決めると $1 - d' \leftarrow \mathcal{D}_1(st, c)$ は $d' \leftarrow \mathcal{A}_1(st, c)$ と表現でき、$\mathcal{D}_0, \mathcal{A}_0$ は式 (4.6) の確率に影響を与えないので、

$$(4.6) = \left| \Pr \left[1 - d' = d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_{1-d}), d' \leftarrow \mathcal{A}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \tag{4.7}$$

$d \leftarrow \$\{0,1\}$ と $1 - d \leftarrow \$\{0,1\}$ 同じ分布なので置き換えることができ、$1 - d' = d \rightarrow d' = 1 - d$ であるので、

$$(4.7) = \left| \Pr \left[d' = 1 - d \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ 1 - d \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_{1-d}), d' \leftarrow \mathcal{A}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \tag{4.8}$$

ここで、$1 - d = \tilde{d}$ とおくと、

$$(4.8) = \left| \Pr \left[d' = \tilde{d} \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), (v_0, v_1, st) \leftarrow \mathcal{A}_0(pk), \\ \tilde{d} \leftarrow \$\{0,1\}, c \leftarrow \mathrm{Enc}(pk, v_{\tilde{d}}), d' \leftarrow \mathcal{A}_1(st, c) \end{array} \right] - \frac{1}{2} \right| \tag{4.9}$$

となり、これは式 (3.3) の左辺と同じ形であるので、

$$(4.9) > \epsilon(k)$$

となり、$\mathcal{A}_0$ の出力が $v_d = 1 - d$ の場合も証明できた。 □

以上, 定理 3.2.1 と定理 3.2.2 が証明できた。

第 5 章

IND-1bitの別表現

本研究では、IND-1bit の安全性を計算機実験で評価するために、補題 5.0.1 を用意する。まず、補題 5.0.1 におけるゲームを定義する。

定義 5.0.1 (補題 5.0.1 におけるゲーム)
ある暗号方式 $\mathcal{E}$ において、攻撃者と挑戦者間で以下のようなゲームを考える。これを補題 *5.0.1* におけるゲームという。

1. 攻撃者は挑戦者から pk を受け取る。
2. 挑戦者は平文 $v = 1$ とし、暗号文 $c = \mathrm{Enc}(pk, 1)$ を攻撃者にわたす。
3. 攻撃者は c から v を予想し v' として出力する。
4. 平文 $v = 0$ のときも同様に *(2)-(3)* を行う。

また、簡単のため確率変数 Succ を定義する。

定義 5.0.2 (確率変数 Succ)
補題 *5.0.1* におけるゲームにおいて、平文 v と攻撃者 $\mathcal{F}$ の出力によって確率変数 Succ を以下のように定義する。

$$\begin{cases} \mathrm{Succ}(v) = 1 : \mathcal{F} \text{ が } v \text{ を出力する} \\ \mathrm{Succ}(v) = 0 : \mathcal{F} \text{ が } 1 - v \text{ を出力する} \end{cases} \tag{5.1}$$

補題 5.0.1
ある暗号方式 $\mathcal{E}$ の攻撃アルゴリズムを $\mathcal{F}$ とする。ある $\mathcal{F}$ が以下を満たすとき、

$$|\Pr[\mathrm{Succ}(1) = 1] - \Pr[\mathrm{Succ}(0) = 0]| > 2\epsilon(k) \tag{5.2}$$

IND-1bit ゲームにおいてある攻撃者 $\mathcal{D}$ が以下を満たす。

$$\left| \Pr\left[v' = v \;\middle|\; \begin{array}{r} (pk, sk) \leftarrow \mathrm{Keygen}(1^k), v \leftarrow \$\{0,1\}, \\ c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(pk, c) \end{array} \right] - \frac{1}{2} \right| > \epsilon(k) \tag{3.2}$$

証明 5.0.1
式 (3.2) の左辺を変形する。$v \leftarrow \$\{0,1\}$ より、 $v' = v$ は $v' = 1$ と $v' = 0$ の *2* つの場合に場合分けできる。また、$v = 0$ と $v = 1$ の場合をランダムにとるので、 $\Pr[v' = v] = \frac{1}{2}\Pr[v' = 1] + \frac{1}{2}\Pr[v' = 0]$ である。

$$
\begin{aligned}
&\left|\Pr\left[v' = v \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), v \leftarrow \$\{0,1\}, \\ c \leftarrow \mathrm{Enc}(pk, v), v' \leftarrow \mathcal{D}(pk, c)\end{array}\right] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}\Pr\left[v' = 1 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 1), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right]\right. \\
&\quad \left. + \frac{1}{2}\Pr\left[v' = 0 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 0), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}\Pr\left[v' = 1 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 1), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right]\right. \\
&\quad \left. + \frac{1}{2}\left(1 - \Pr\left[v' \neq 0 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 0), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right]\right) - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}\Pr\left[v' = 1 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 1), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right]\right. \\
&\quad \left. + \frac{1}{2} - \frac{1}{2}\Pr\left[v' \neq 0 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 0), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right] - \frac{1}{2}\right| \\
&= \frac{1}{2}\left|\Pr\left[v' = 1 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 1), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right]\right. \\
&\quad \left. - \Pr\left[v' \neq 0 \;\middle|\; \begin{array}{r}(pk, sk) \leftarrow \mathrm{Keygen}(1^k), \\ c \leftarrow \mathrm{Enc}(pk, 0), v' \leftarrow \mathcal{F}(pk, c)\end{array}\right]\right| \\
&> \frac{1}{2} 2\epsilon(k) = \epsilon(k)
\end{aligned}
$$

より式 (5.2) から式 (3.2) が導ける。よって、補題 *5.0.1* が証明できた。

第 6 章

攻撃手法

本章では、LWE暗号での補題5.0.1における攻撃者$\mathcal{F}$を作成した。具体的にはCompact-LWEに対する攻撃手法 [2] を、LWE暗号に適応できるよう改良した。

6.1 Compact-LWEに対する攻撃手法 [2]

6.1.1 方針

暗号文$c = (c_0, c_1)$と公開鍵$pk = \{\vec{\alpha_i}, \beta_i\}_{1 \leq i \leq m}$から以下のような$M$をつくる。ただし$A = (\vec{\alpha_1}^\top, \ldots, \vec{\alpha_m}^\top)^\top$, $\vec{\beta} = (\beta_1, \ldots, \beta_m)$とする。

$$M(pp, pk, c_0, c_1) = \begin{pmatrix} 1 & O_{1\times m} & \kappa c_0 & c_1 \\ O_{m\times 1} & tI_m & -\kappa A & \vec{\beta}^\top \\ 0 & O_{1\times m} & O_{1\times n} & q \end{pmatrix} \tag{6.1}$$

ここでqはCompact-LWEのパラメータ、I_mは$m \times m$の単位行列、κは適当に大きな定数で今回は$\kappa = q$とする。

6.1.2 アルゴリズム

pp,*pk*,*c*を入力し、vを出力する。

1. $\kappa = q$とし、$M = M(pp, pk, c_0, c_1)$を求める。
2. Mの行ベクトルがつくる格子の簡約化基底$\{\vec{\mu_i}\}_{1 \leq i \leq m}$を求める。 ここで簡約化基底は小さい順に並んでいる。
3. $1 \leq i \leq m$で$\vec{\mu_i}$の最初の項をu_i、最終の項をv_iとし、はじめて$u_i \neq 0$のときのiをjとする。
4. $1 < i < j$までのv_iの最大公約数gを求める。
5. もし$g \geq t$なら, $v_j \bmod g$を出力。それ以外ならv_jを出力。

6.2　新たなLWE暗号の安全性評価手法

6.2.1　方針

6.1 章の Compact-LWE に対する攻撃方法を LWE 暗号に対して使えるようにするため、式 (6.1) を以下のように変更した。ただし $A = (\vec{a_1}^\top, \ldots, \vec{a_m}^\top)^\top$, $\vec{\beta} = (b_1, \ldots, b_m)$ とする。

$$M_{\mathrm{LWE}}(pp, pk, c_0, c_1) = \begin{pmatrix} 1 & O_{1\times m} & \kappa c_0 & c_1 \\ O_{m\times 1} & tI_m & -\kappa A & -\vec{\beta}^T \end{pmatrix} \tag{6.2}$$

6.2.2　M_{LWE} を適応したアルゴリズム $\mathcal{F}$

pp,*pk*,*c* を入力し、v を出力する。

1. $\kappa = q$ とし、$M_{\mathrm{LWE}} = M_{\mathrm{LWE}}(pp, pk, c_0, c_1)$ を求める。
2. M_{LWE} の行ベクトルがつくる格子の簡約化基底 $\{\vec{\mu_i}\}_{1\le i\le m}$ を求める。 ここで簡約化基底は小さい順に並んでいる。
3. $1 \le i \le m$ で $\vec{\mu_i}$ の最初の項を u_i、最終の項を v_i とし、はじめて $u_i \neq 0$ のときの i を j とする。
4. $v_i \bmod q$ が $\left\lfloor \frac{q}{2} \right\rfloor$ より 0 に近い値をとれば $v = 0$。 それ以外ならば $v = 1$ を出力。

第 7 章

実験

6.2 章の手法をもとに実験を行った。

7.1 方法

1. KeyGen(pp) を実行し、(sk, pk) を一組求める。
2. 平文 $v = 1$ として, Enc(pk, v) を実行し、暗号文 c を得る。
3. 6.2 章のアルゴリズム $\mathcal{F}$ を実行し、v' を得る。
4. (2)-(3) を 1000 回繰り返し、確率 $\Pr[\mathrm{Succ}(1) = 1]$ を得る。
5. 平文 $v = 0$ に対しても (2)-(4) を同様に行い確率 $\Pr[\mathrm{Succ}(0) = 1]$ を得る。
6. $|\Pr[\mathrm{Succ}(1) = 1] - (1 - \Pr[\mathrm{Succ}(0) = 1])| = |\Pr[\mathrm{Succ}(1) = 1] - \Pr[\mathrm{Succ}(0) = 0]|$ を出力する。
7. (1)-(6) を 1000 回繰り返す。

この実験では、30 ビット安全性を保証可能なパラメータ pp を lwe-estimator[5] を用いて導出した。

表 7.1: 各パラメータの値

n	q	m	σ
20	401	10	4.79997

7.1.1 ソースコード

文献 [2] に示してある SageMath を用いたプログラムを改良し、アルゴリズム $\mathcal{F}$ に基づいたプログラム (SageMath) を作成した。

ソースコード 7.1: アルゴリズム $\mathcal{F}$ に基づいたプログラム

```
# Make the experiment reproducible
# ( at least on given platform / Sage version )
from time import time
set_random_seed (time())

# cputime
timecpu = cputime()

# LWE parameters
n = 20
m = 10
t = 2
q = 401
l = 1
r = 1

sigma = q * 0.01197
from sage.stats.distributions.discrete_gaussian_integer import
  DiscreteGaussianDistributionIntegerSampler
D = DiscreteGaussianDistributionIntegerSampler(sigma=sigma)

ppro = [0.5, 0.5]
U = GeneralDiscreteDistribution(ppro)
# ========================

def keygen ():
    s = random_matrix ( ZZ ,n ,l , x =0 , y = q )
    return s

def samplegen ( s ):
    A = random_matrix ( ZZ ,m ,n , x =0 , y = q )
    e = matrix ([vector ([ D() for _ in range ( l )])for _ in range ( m )])
    v = (A*s)%q +e
    return A , v%q

def encrypt (A ,v , mu ):
    u = vector ([ U.get_random_element() for _ in range ( m )])

    a = A.transpose() * u
    x = v.transpose() * u + vector ([ mu * floor(q / t) for _ in range ( l ) ])
    return a%q , x%q

def decrypy ( s ,a ,x ):
    ifcand = 0
    for j in range ( l ):
        candx = x[j] - ((s.transpose()[j] * a)%q)
        if (candx % q) >= round(q / 4) and (candx % q) <= round(3*q / 4):
            ifcand = 1
    return ifcand

# # ========================

def subsetsumdecrypt (A ,v ,a , x ):
    kappa = q
    ifcand = 0
    cand = vector ( ZZ , l )
    for j in range ( l ):
        L = block_matrix ( ZZ , \
            [[1 , 0 ,                          kappa * a.row () , x [ j ] ] , \
            [0 , t*identity_matrix ( m ) , - kappa * A ,-v.column (j).column ()]])
        L = L.LLL ()
```

```
61
62          # index of first non - zero entry in the first column of L
63          idx = next (( i for i , x [ j ] in enumerate ( L.column (0). list ()) if x [ j ]
                !=0))
64          candx = L [ idx , -1]/ L [ idx ,0]
65          if (candx % q) >= round(q / 4) and (candx % q) <= round(3*q / 4):
66              ifcand = 1
67      return ifcand
68
69  # # ========================
70
71  def testsubsetsumdecrypt ( trials ):
72
73      # key product
74      s = keygen ()
75      A ,v = samplegen ( s )
76
77      # Plaintext = 1
78      succmucand = 0
79      succdec = 0
80      for _ in range ( trials ):
81          mu = 1
82          a , x = encrypt (A ,v , mu )
83
84          # Decryption
85          mudec = decrypy ( s ,a ,x )
86          if mu == mudec:
87              succdec += 1
88          # Attack
89          mucand = subsetsumdecrypt (A ,v ,a , x )
90          if mu == mucand:
91              succmucand += 1
92
93      attack1 = float( 100 * succmucand / trials )
94      decpro1 = float( 100 * succdec / trials )
95
96      # Plaintext = 0
97      succdec = 0
98      succmucand = 0
99      for _ in range ( trials ):
100         mu = 0
101         a , x = encrypt (A ,v , mu )
102
103         # Decryption
104         mudec = decrypy ( s ,a ,x )
105         if mu == mudec:
106             succdec += 1
107         # Attack
108         mucand = subsetsumdecrypt (A ,v ,a , x )
109         if mu == mucand:
110             succmucand += 1
111
112     attack0 = float( 100 * succmucand / trials )
113     decpro0 = float( 100 * succdec / trials )
114
115     return attack1, attack0,decpro1, decpro0
116
117 attack1, attack0,decpro1, decpro0 = testsubsetsumdecrypt ( trials = 1000 )
118 #output
119 print attack1, attack0, attack1 - (100 - attack0 ), decpro1, decpro0, cputime(timecpu)
```

また、SageMath をターミナル上で実行するためのプログラム (シェルスクリプト) と、プ

ログラムを並列で繰り返すためのプログラム (シェルスクリプト) を作成した。

ソースコード 7.2: SageMath をターミナル上で実行するためのプログラム

```
#!/bin/bash
# sageファイルをnum回実行 , ファイルに回数とともに書き込み
i=1
while [ $i -le $num ]
do
    echo -n `expr␣$numsum␣+␣$i␣` "">> $file.dat | sage ${file%-*}.sage >> $file.dat
    # エラー処理
    if [ $? != 0 ]; then
        echo "error" >> $file.dat
    fi
    i=`expr␣$i␣+␣1`
done
```

ソースコード 7.3: プログラムを並列で繰り返すためのプログラム

```
#!/bin/bash

#--標準入力処理--
# sagefile名入力
while :
do
    # sagefile名
    read -p "sageファイル名:␣" -e filename # -e tab補完 -p メッセージ

    if [[ -e $filename ]]; then # -e 存在するか
        if [[ ${filename##*.} = "sage" ]]; then
            break
        fi
        echo error: sageファイルではありません.
    else
        echo error: sageファイルが存在しません.
    fi
done

# 繰り返し回数, 分割数入力
while :
do
    # 繰り返し回数
    read -p "繰り返し回数:␣" nums
    # 分割数
    read -p "並列数:␣" div
    # 繰り返し回数<分割数
    if [ $nums -ge $div ]; then
        break
    fi
    echo error: 並列数が繰り返し回数より多いです.
done

# パラメータの有無
read -p "パラメータの有無(y/n):␣" paramater
# 要素名の有無
read -p "要素名の有無(y/n):␣" element
# csvを出力するか
read -p "csvを出力する?(y/n):␣" csv

#----

```

```
#--同じ名前のdateファイルを上書きしない処理 --
i=1
while :
do
    filecom=${filename%.sage}"-"$i
    # ファイルが存在するか確認
    RES=`find␣$filecom*.dat␣2>/dev/null`
    if [[ -z "$RES" ]]; then
        echo -n "出力ファイル:␣"
        echo $filecom
        break
    fi

    i=`expr␣$i␣+␣1`
done

#----

#--並列処理--
numq=`expr␣$nums␣/␣$div`
numr=`expr␣$nums␣%␣$div`
numi=1
numsum=0
# 余りがある時の処理
while [ $numi -le $numr ]
do
    file=$filecom"_"$numi
    # 繰り返し回数を+1
    num=`expr␣$numq␣+␣1`
    # file名, 回数, 表示回数をexport
    export file
    export num
    export numsum
    # シェルスクリプトを実行
    sagerepeat.sh &
    sleep 1
    # 回数を更新
    numi=`expr␣$numi␣+␣1`
    numsum=`expr␣$numsum␣+␣$num␣`
done

# 余りがないor余り以外の処理
while [ $numi -le $div ]
do
    file=$filecom"_"$numi
    num=$numq
    # file名, 回数, 表示回数をexport
    export file
    export num
    export numsum
    # シェルスクリプトを実行
    sagerepeat.sh &
    sleep 1
    #回数を更新
    numi=`expr␣$numi␣+␣1`
    numsum=`expr␣$numsum␣+␣$num␣`
done

# 処理が終わるまでwait
wait

# 並列での結果を1つのファイルにまとめる
```

```
105
106 # ヘッダー
107 # 回数
108 echo "全体の繰り返し回数:␣$nums" >> $filecom.dat
109 echo -n "プログラムの繰り返し回数:␣" >> $filecom.dat
110 grep "trials␣=␣"  $filename | tail -n 1 | sed -e 's/.* trials =//g'| sed -e 's/)//g' |
      sed -e 's/ //g'>> $filecom.dat
111 echo >> $filecom.dat
112 # パラメータ
113 if [ $paramater = "y" ]; then
114     echo "#"--LWE parameters--  >> $filecom.dat
115     grep -m1 "n␣="  $filename >> $filecom.dat
116     grep -m1 "m␣="  $filename >> $filecom.dat
117     grep -m1 "q␣="  $filename >> $filecom.dat
118     grep -m1 "sigma␣="  $filename >> $filecom.dat
119     grep -m1 "t␣="  $filename >> $filecom.dat
120     echo "#"----  >> $filecom.dat
121     echo >> $filecom.dat
122 fi
123 # 実行結果
124 # 要素名
125 if [ $element = "y" ]; then
126     echo -n "num␣" >> $filecom.dat
127     grep "print"  $filename | tail -n 1 | sed -e 's/print //g'| sed -e 's/,//g' >>
        $filecom.dat
128 fi
129 # 結合
130 i=1
131 while [ $i -le $div ]
132 do
133     cat $filecom"_"$i.dat >> $filecom.dat
134     rm $filecom"_"$i.dat
135     i=`expr␣$i␣+␣1`
136 done
137
138 #----
139
140 #--csvファイルを出力する --
141 if [ $csv = "y" ]; then
142     tail -n +`sed␣'/^1␣/q'␣$filecom.dat␣|␣wc␣-l` $filecom.dat | sed -e 's/ /,/g' >
        $filecom.csv
143 fi
144
145 #----
```

7.2　結果と考察

5 回実験した。結果は以下のようになった。

表 7.2: 結果

	1 回目	2 回目	3 回目	4 回目	5 回目
$\lvert \Pr[\mathrm{Succ}(1)=1] - \Pr[\mathrm{Succ}(0)=0] \rvert$ の平均 $(\times 10^{-2})$	3.2862	3.3864	3.2866	3.1351	3.4309

実験結果から LWE 暗号の公開鍵から秘密鍵を取り出すための計算時間を求める。[6] によると LWE 暗号の公開鍵から秘密鍵を取り出すための計算時間は、$\mathcal{O}\left(nq\frac{\log nq}{\epsilon(k)}\right)\times T$ で求められる。ここで T は、簡約化基底を求めるアルゴリズムの計算時間である。補題 5.0.1 より、実験結果は $|\Pr[\mathrm{Succ}(1)=1]-\Pr[\mathrm{Succ}(0)=0]|>2\epsilon(k)$ (式 (5.2)) の $2\epsilon(k)$ の平均を表している。実験結果は $3.1351\times10^{-2}\sim3.4309\times10^{-2}$ であるので, 計算の簡単のため $2\epsilon(k)=3.2000\times10^{-2}\rightarrow\epsilon(k)=1.6000\times10^{-2}$ として計算時間すると以下のようになる.

$$\begin{aligned}\mathcal{O}\left(nq\frac{\log nq}{\epsilon(k)}\right)\times T &\approx \mathcal{O}\left(20\times401\frac{\log 20\times401}{\frac{16}{1000}}\right)\times T\\ &\approx \mathcal{O}\left(2^{23}\right)\times T\end{aligned}$$

文献 [5] によると簡約化基底を求めるアルゴリズムの計算時間は、6.2 章での行列 $M_{LWE}(6.2)$ の行数を N、列数を M、一つの行のノルムを B とすると $\mathcal{O}\left(N^3\log^2 B\right)$ で求められる。また、$N=1+m=11, B=\sqrt{M\times q^2}=\sqrt{(n+m+2)\times q^2}\approx2300$ である。上記の式に、この方法で導出した T を代入すると以下のようになる。

$$\begin{aligned}\mathcal{O}\left(2^{23}\right)\times\mathcal{O}\left(N^3\log^2 B\right) &\approx \mathcal{O}\left(2^{23}\right)\times\mathcal{O}\left(11^3\log^2 2300\right)\\ &\approx \mathcal{O}\left(2^{23}\right)\times\mathcal{O}\left(2^{17}\right)\approx\mathcal{O}\left(2^{40}\right)\end{aligned}$$

本実験では 30 ビット安全を保証可能なパラメータを用いたため、想定されている LWE 暗号の公開鍵から秘密鍵を取り出すための計算時間は $\mathcal{O}\left(2^{30}\right)$ である。実験結果より導出した計算時間は $\mathcal{O}\left(2^{40}\right)$ であるため、想定されている攻撃者よりも非効率的であることがわかる。従って、30 ビット安全性を想定した lwe-estimator[5] によって設定されるパラメータは我々の提案攻撃に対して安全であるといえる。

第 8 章

追加実験

31 ビット安全性、32 ビット安全性を想定した lwe-estimator[5] によって設定されるパラメータでも 7 章と同様に実験を行った。

8.1 パラメータ

各ビット安全性を想定した lwe-estimator[5] によって設定されるパラメータは以下である。

表 8.1: 31 ビット安全性を想定したパラメータの値

n	q	m	σ
32	1031	21	7.29027

表 8.2: 32 ビット安全性を想定したパラメータの値

n	q	m	σ
54	2917	51	11.98577

8.2 結果と考察

各パラメータにつき、5 回実験した。結果は以下のようになった。

表 8.3: 31 ビット安全性を想定したパラメータの実験結果

	1 回目	2 回目	3 回目	4 回目	5 回目
$\lvert\Pr[\mathrm{Succ}(1)=1]-\Pr[\mathrm{Succ}(0)=0]\rvert$ の平均 $(\times 10^{-2})$	1.7313	1.7830	1.7885	1.7648	1.7676

表 8.4: 32 ビット安全性を想定したパラメータの実験結果

	1 回目	2 回目	3 回目	4 回目	5 回目
$\lvert\Pr[\mathrm{Succ}(1)=1]-\Pr[\mathrm{Succ}(0)=0]\rvert$ の平均 $(\times 10^{-2})$	1.8852	1.8672	1.7584	1.8213	1.7575

7 章と同様に、実験結果から LWE 暗号の公開鍵から秘密鍵を取り出すための計算時間を求めると以下のようになった。

表 8.5: LWE 暗号の公開鍵から秘密鍵を取り出すための計算時間

31 ビット	32 ビット
$\mathcal{O}(2^{47})$	$\mathcal{O}(2^{54})$

両パラメータにおける計算量とも lwe-estimator[5] が想定した計算量より大きくなった。よって、31 ビット安全性、32 ビット安全性を想定した lwe-estimator[5] によって設定されるパラメータは我々の提案攻撃に対して安全であるといえる。

このような結果になった要因として、我々の提案攻撃で用いた行列 M_{LWE} が lwe-estimator[5] が用いているよりも非効率である可能性が指摘できる。今後の課題として、行列 M_{LWE} の効率性を調べ、より精度が高い攻撃となるよう改良していくことが挙げられる。

また、IND-CPA 安全の $\epsilon(k)$ から LWE 暗号の公開鍵から秘密鍵を取り出すための計算時間への帰着について文献 [6] を参考にしたが、正確に評価していない可能性がある。今後の課題として、IND-CPA 安全の $\epsilon(k)$ から LWE 暗号の公開鍵から秘密鍵を取り出すための計算時間への帰着を文献 [6] より正確に証明し、計算時間の正確な評価を行いたい。

謝辞

本研究は電気通信大学准教授バグスサントソ先生のご指導のもとで、筆者が電気通信大学情報理工学研究科在学中に行った LWE 暗号における IND-CPA 安全性の再評価について研究成果をまとめたものである。本研究を進めるにあたり先生から賜ったご指導、ご鞭撻に対し深く感謝いたします。また、大濱先生や大濱研の皆様、バグス研の皆様、特に同期の竹牟禮 薫さんにも日頃からのご助言深く感謝いたします。

参考文献

[1] Dongxi Liu, Nan Li, Jongkil Kim, and Surya Nepal. Compact-lwe: Enabling practically lightweight public key encryption for leveled iot device authentication. Cryptology ePrint Archive, Report 2017/685, 2017.

[2] Jonathan Bootle, Mehdi Tibouchi, and Keita Xagawa. Cryptanalysis of compact-lwe. In *CT-RSA 2018*, pages 80–97, 2018.

[3] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *arXiv e-prints*, pages quant–ph/9508027, August 1995.

[4] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. ACM.

[5] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169 – 203, 01 Oct. 2015.

[6] Britt Cyr and Vinod Vaikuntanathan. Lecture2 - 6.892 computing on encrypted data, 2013. `https://people.csail.mit.edu/vinodv/6892-Fall2013/lecture02.pdf`.